Sequence Alignment

Outline

- Global Alignment
- Scoring Matrices
- Local Alignment
- Alignment with Affine Gap Penalties

From LCS to Alignment: Change up the Scoring

- The Longest Common Subsequence (LCS) problem —the simplest form of sequence alignment – allows only insertions and deletions (no mismatches).
- In the LCS Problem, we scored 1 for matches and 0 for indels
- Consider penalizing indels and mismatches with negative scores
- Simplest *scoring schema*:

  $+1$: **match premium**

  $-\mu$: **mismatch penalty**

  $-\sigma$: **indel penalty**

Simple Scoring

- When mismatches are penalized by –μ, indels are penalized by –σ,

  and matches are rewarded with *+1*,

  the resulting score is:

$$\#matches - \mu(\#mismatches) - \sigma(\#indels)$$

The Global Alignment Problem

Find the best alignment between two strings under a given scoring schema

Input : Strings **v** and **w** and a scoring schema
Output : Alignment of maximum score

↑ → = -б
↘ = *1* if match
= -*μ* if mismatch

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + 1 & \text{if } v_i = w_j \\ s_{i-1,j-1} - \mu & \text{if } v_i \neq w_j \\ s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \end{cases}$$

m : mismatch penalty
σ : indel penalty

Scoring Matrices

To generalize scoring, consider a (4+1) x(4+1) **scoring matrix** δ.

In the case of an amino acid sequence alignment, the scoring matrix would be a (20+1)x(20+1) size. The addition of 1 is to include the score for comparison of a gap character "-".

This will simplify the algorithm as follows:

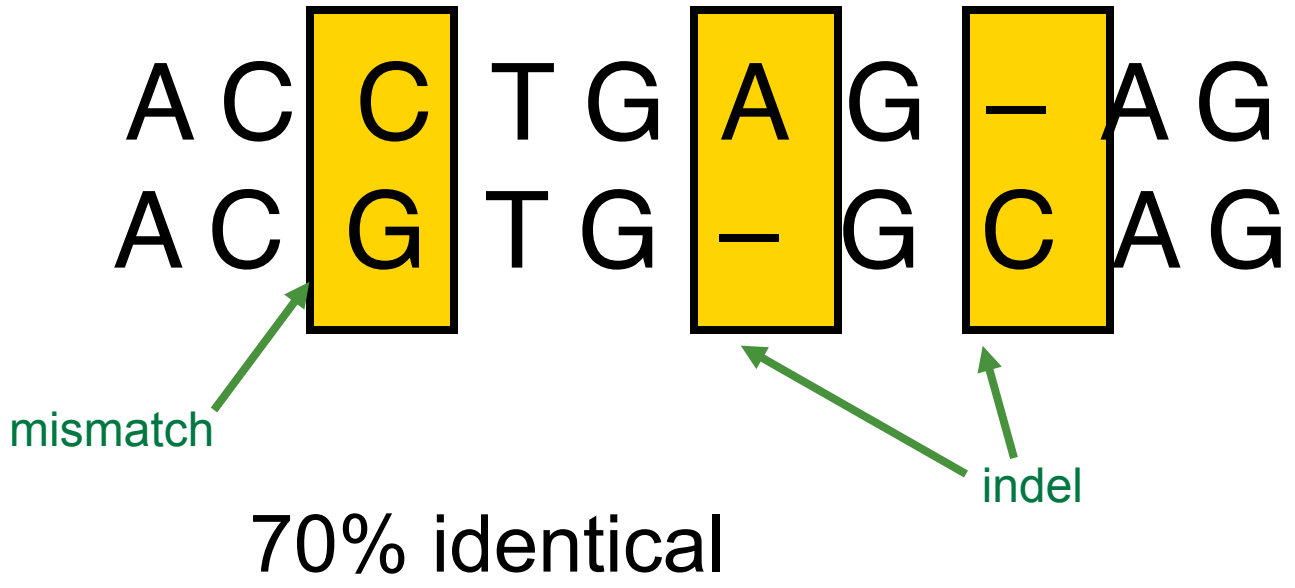
$$s_{i,j} = max \begin{cases} s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{cases}$$

Measuring Similarity

- Measuring the extent of similarity between two sequences
  - Based on percent sequence identity
  - Based on conservation

Percent Sequence Identity

- The extent to which two nucleotide or amino acid sequences are invariant



Making a Scoring Matrix

- Scoring matrices are created based on biological evidence.
- Alignments can be thought of as two sequences that differ due to mutations.
- Some of these mutations have little effect on the protein's function, therefore some penalties, $\delta(v_i, w_j)$, will be less harsh than others.

Scoring Matrix: Example

	A	R	N	K
A	5	-2	-1	-1
R	-	7	-1	3
N	-	-	7	0
K	-	-	-	6

AKRANR
KAAANK

- Notice that although R and K are different amino acids, they have a positive score.
- Why? They are both positively charged amino acidsà will not greatly change function of protein.

-1 + (-1) + (-2) + 5 + 7 +

Conservation

- Amino acid changes that tend to preserve the physico-chemical properties of the original residue
  - Polar to polar
    - aspartate à glutamate
  - Nonpolar to nonpolar
    - alanine à valine
  - Similarly behaving residues
    - leucine to isoleucine

Scoring matrices

- Amino acid substitution matrices
  - PAM
  - BLOSUM
- DNA substitution matrices
  - DNA is less conserved than protein sequences
  - Less effective to compare coding regions at nucleotide level

PAM

- **P**oint **A**ccepted **M**utation (Dayhoff et al.)
- 1 PAM = PAM_1 = 1% average change of all amino acid positions
  - After 100 PAMs of evolution, not every residue will have changed
    - some residues may have mutated several times
    - some residues may have returned to their original state
    - some residues may not changed at all

PAM_X

- $PAM_x = PAM_1^x$
  - $PAM_{250} = PAM_1^{250}$
- PAM_{250} is a widely used scoring matrix:

		Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	...
		A	R	N	D	C	Q	E	G	H	I	L	K	...
Ala	A	13	6	9	9	5	8	9	12	6	8	6	7	...
Arg	R	3	17	4	3	2	5	3	2	6	3	2	9	
Asn	N	4	4	6	7	2	5	6	4	6	3	2	5	
Asp	D	5	4	8	11	1	7	10	5	6	3	2	5	
Cys	C	2	1	1	1	52	1	1	2	2	2	1	1	
Gln	Q	3	5	5	6	1	10	7	3	7	2	3	5	
...														
Trp	W	0	2	0	0	0	0	0	0	1	0	1	0	
Tyr	Y	1	1	2	1	3	1	1	1	3	2	2	1	
Val	V	7	4	4	4	4	4	4	4	5	4	15	10	

BLOSUM

- **Blo**cks **Su**bstitution **M**atrix
- Scores derived from *observations* of the frequencies of substitutions in blocks of local alignments in related proteins
- Matrix name indicates evolutionary distance
  - BLOSUM62 was created using sequences sharing no more than 62% identity

The Blosum50 Scoring Matrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0	-2	-1	-1	-5
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3	-1	0	-1	-5
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3	4	0	-1	-5
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4	5	1	-1	-5
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1	-3	-3	-2	-5
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3	0	4	-1	-5
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3	1	5	-1	-5
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4	-1	-2	-2	-5
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4	0	0	-1	-5
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4	-4	-3	-1	-5
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1	-4	-3	-1	-5
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3	0	1	-1	-5
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1	-3	-1	-1	-5
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1	-4	-4	-2	-5
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3	-2	-1	-2	-5
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2	0	0	-1	-5
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0	0	-1	0	-5
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3	-5	-2	-3	-5
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1	-3	-2	-1	-5
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5	-4	-3	-1	-5
B	-2	-1	4	5	-3	0	1	-1	0	-4	-4	0	-3	-4	-2	0	0	-5	-3	-4	5	2	-1	-5
Z	-1	0	0	1	-3	4	5	-2	0	-3	-3	1	-1	-4	-1	0	-1	-2	-2	-3	2	5	-1	-5
X	-1	-1	-1	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-1	0	-3	-1	-1	-1	-1	-1	-5
*	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	1

Local vs. Global Alignment

- The Global Alignment Problem tries to find the longest path between vertices *(0,0)* and (*n,m*) in the edit graph.

- The Local Alignment Problem tries to find the longest path among paths between **arbitrary vertices** (*i,j*) and (*i', j'*) in the edit graph.

Local vs. Global Alignment

- The Global Alignment Problem tries to find the longest path between vertices *(0,0)* and (*n,m*) in the edit graph.

- The Local Alignment Problem tries to find the longest path among paths between **arbitrary vertices** (*i,j*) and (*i', j'*) in the edit graph.
- In the edit graph with negatively-scored edges, Local Alignmet may score higher than Global Alignment

Local vs. Global Alignment (cont'd)

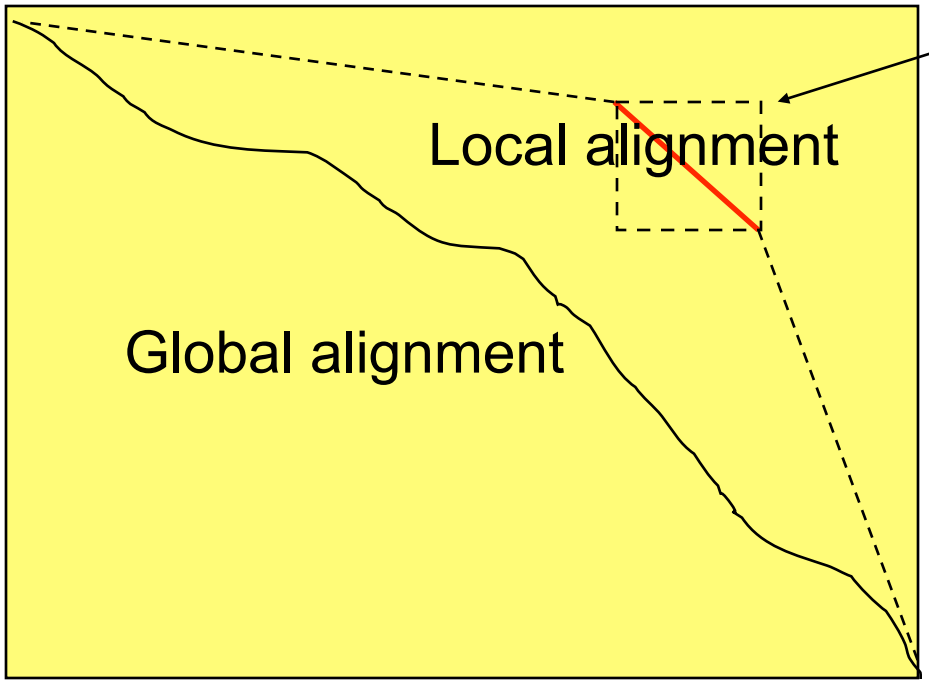
- Global Alignment

```
--T--CC-C-AGT--TATGT-CAGGGGACACG-A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || | | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG-T-CAGAT--C
```

- Local Alignment—better alignment to find conserved segment

```
                  tccCAGTTATGTCAGgggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

Local Alignment: Example



Compute a “mini” Global Alignment to get Local

Local Alignments: Why?

- Two genes in different species may be similar over short conserved regions and dissimilar over remaining regions.
- Example:
  - Homeobox genes have a short region called the *homeodomain* that is highly conserved between species.
  - A global alignment would not find the homeodomain because it would try to align the ENTIRE sequence

The Local Alignment Problem

- Goal: Find the best local alignment between two strings
- Input : Strings **v, w** and scoring matrix δ
- Output : Alignment of substrings of **v** and **w** whose alignment score is maximum among all possible alignment of all possible substrings

The Problem with this Problem

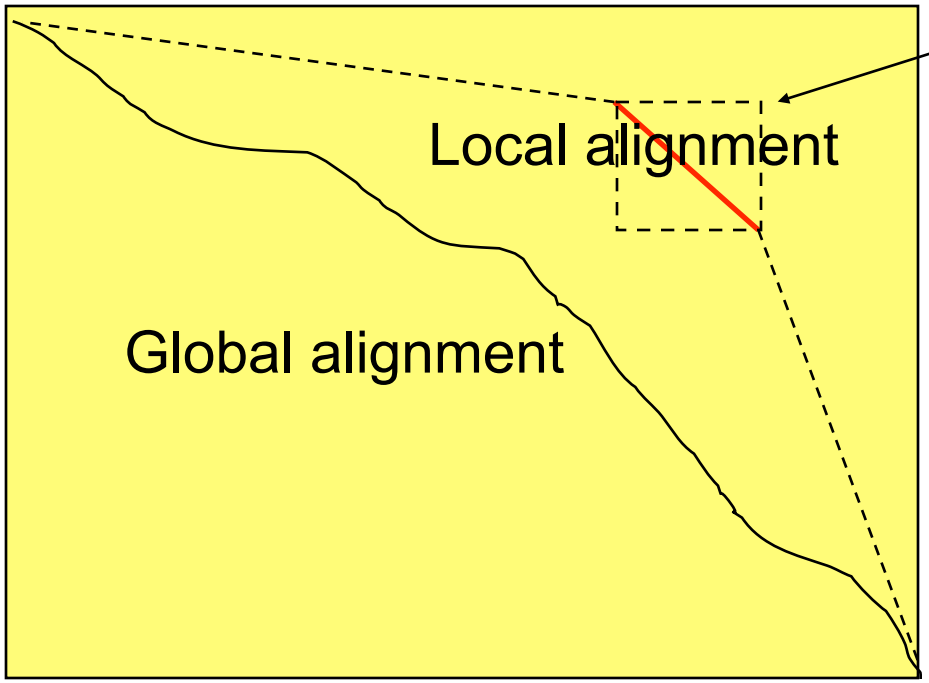
- Long run time $O(n^4)$:

  - In the grid of size $n \times n$ there are $\sim n^2$ vertices (i,j) that may serve as a source.

  - For each such vertex computing alignments from (i,j) to (i',j') takes $O(n^2)$ time.

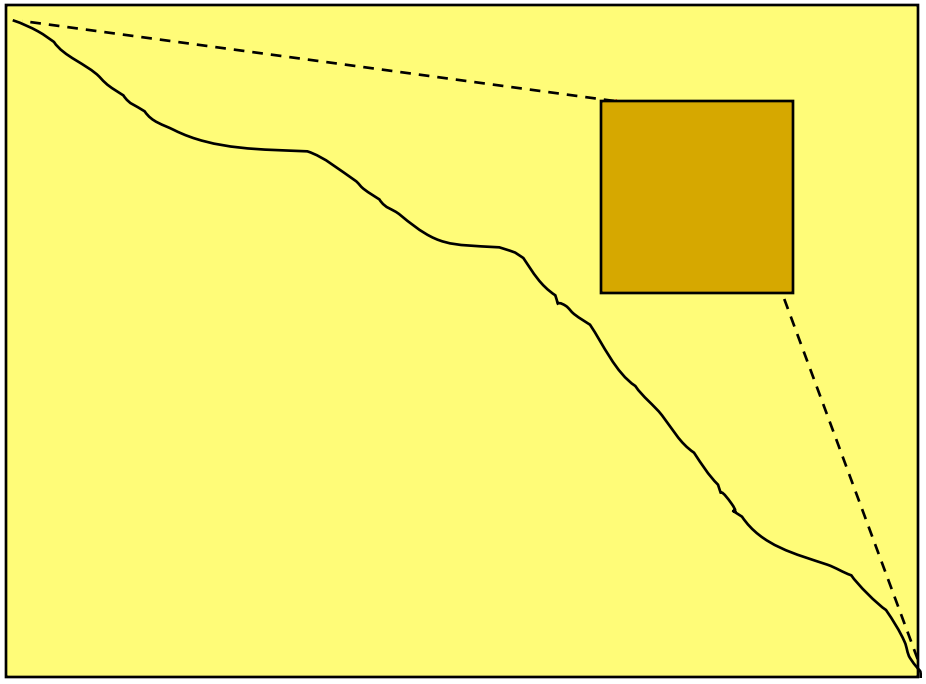
- This can be remedied by giving free rides

Local Alignment: Example

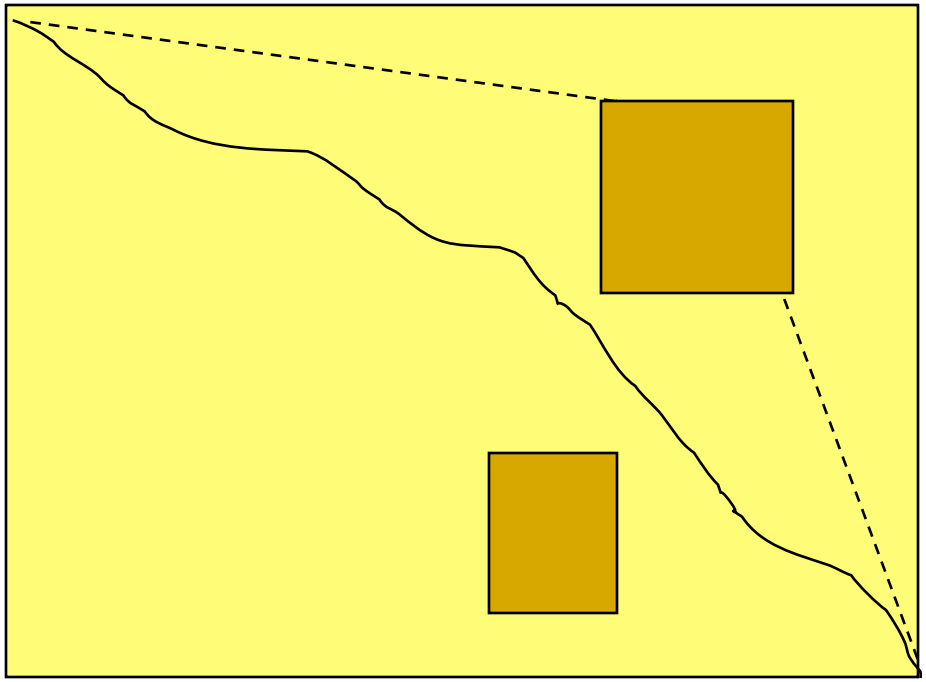


Compute a “mini” Global Alignment to get Local

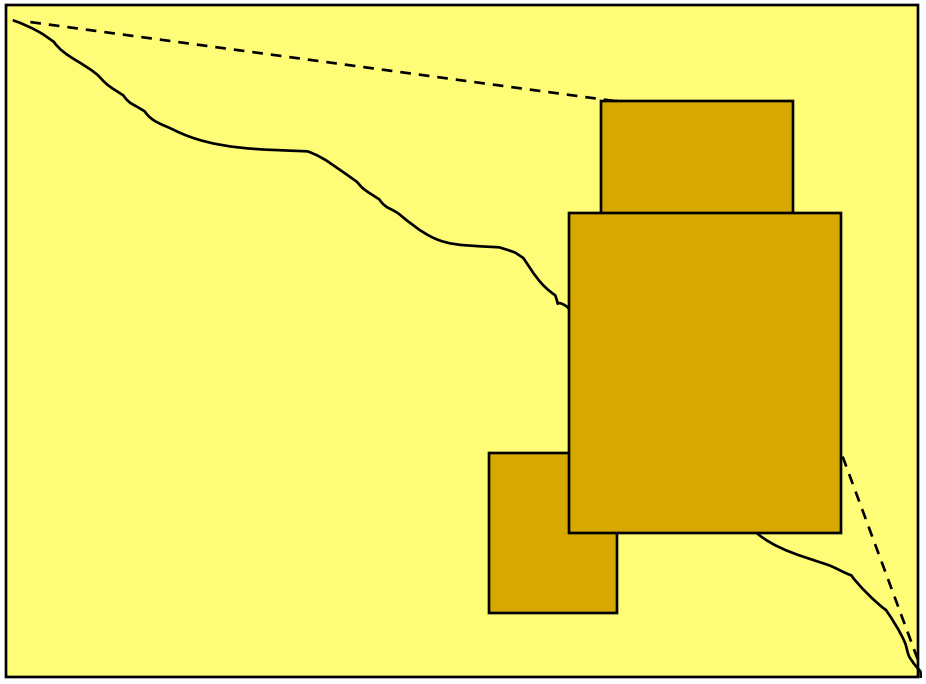
Local Alignment: Example



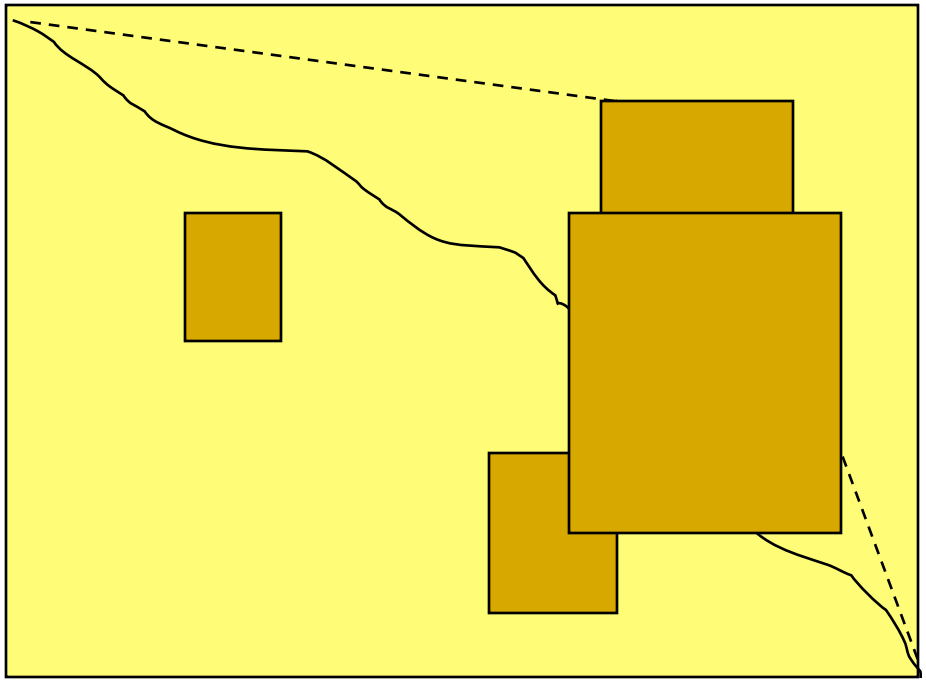
Local Alignment: Example



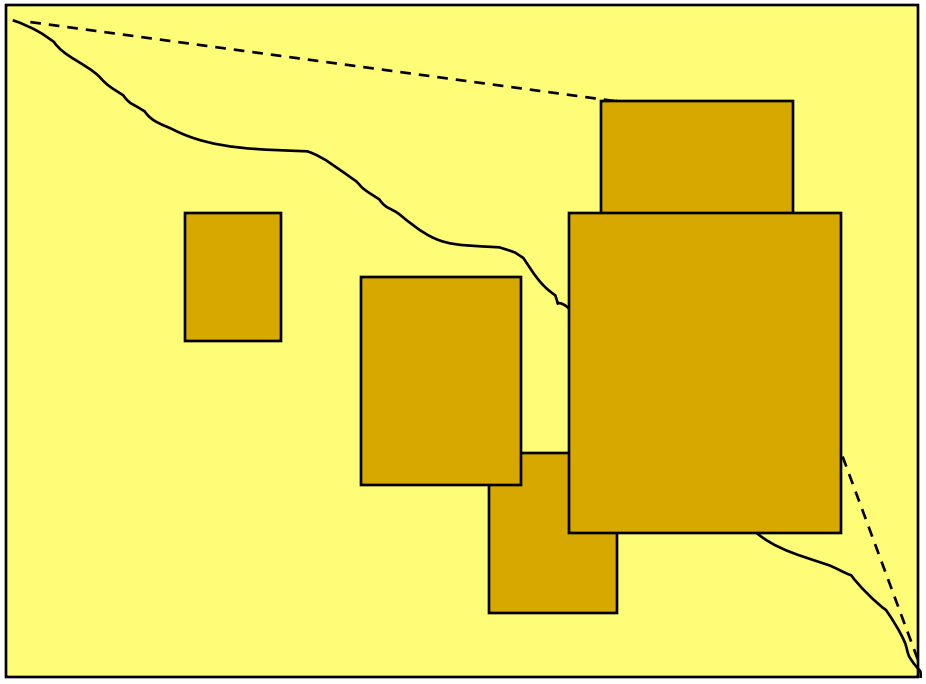
Local Alignment: Example



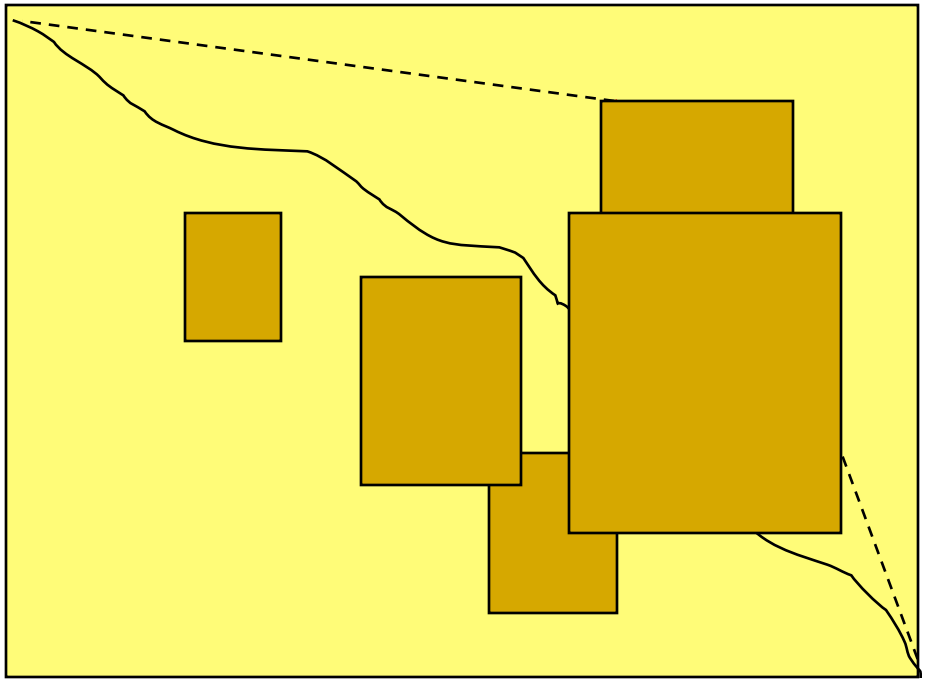
Local Alignment: Example



Local Alignment: Example



Local Alignment: Running Time



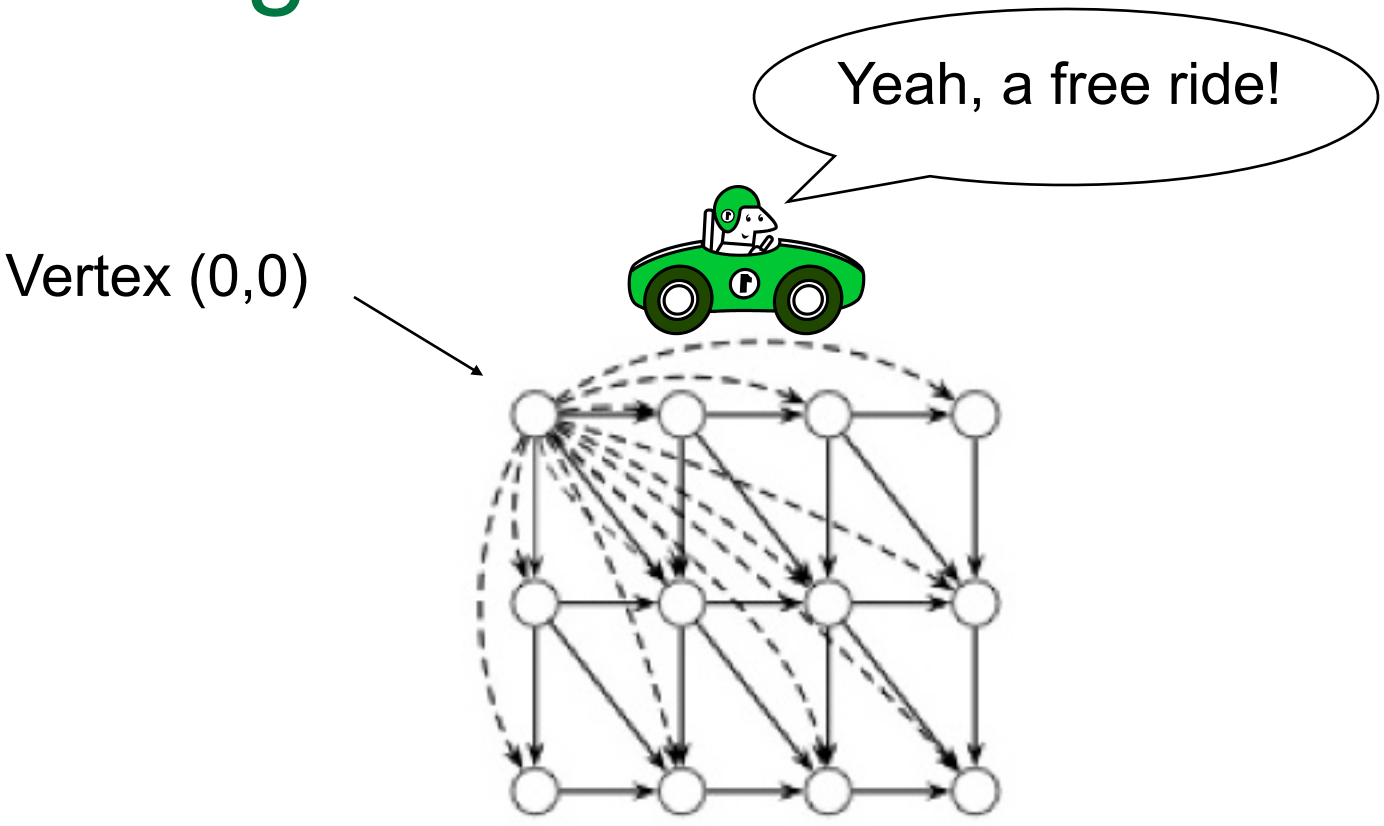
- Long run time $O(n^4)$:

  - In the grid of size $n \times n$ there are $\sim n^2$ vertices (i,j) that may serve as a source.

  - For each such vertex computing alignments from (i,j) to (i',j') takes $O(n^2)$ time.

- This can be remedied by giving free rides

Local Alignment: Free Rides



The dashed edges represent the free rides from (0,0) to every other node.

The Local Alignment Recurrence

- The largest value of $s_{i,j}$ over the whole edit graph is the score of the best local alignment.

- The recurrence:

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{cases}$$

Notice there is only this change from the original recurrence of a Global Alignment

The Local Alignment Recurrence

- The largest value of $s_{i,j}$ over the whole edit graph is the score of the best local alignment.

- The recurrence:

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{cases}$$

Power of ZERO: there is only this change from the original recurrence of a Global Alignment - since there is only one "free ride" edge entering into every vertex

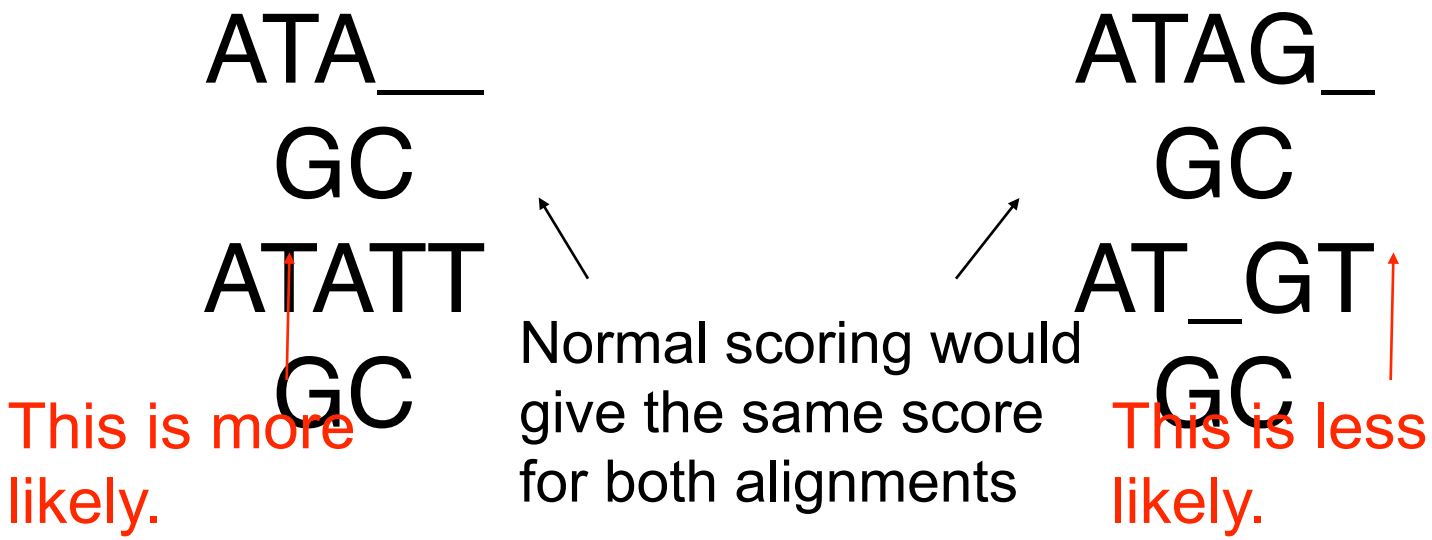
Scoring Indels: Naive Approach

- A fixed penalty σ is given to every indel:
  - -σ for 1 indel,
  - -2σ for 2 consecutive indels
  - -3σ for 3 consecutive indels, etc.

Can be too severe penalty for a series of 100 consecutive indels

Affine Gap Penalties

- In nature, a series of *k* indels often come as a single event rather than a series of *k* single nucleotide events:



Accounting for Gaps

- *Gaps*- contiguous sequence of spaces in one of the rows

- Score for a gap of length x is:

$$-(\rho + \sigma x)$$

where $\rho > 0$ is the penalty for introducing a gap:

gap opening penalty

ρ will be large relative to σ*:*

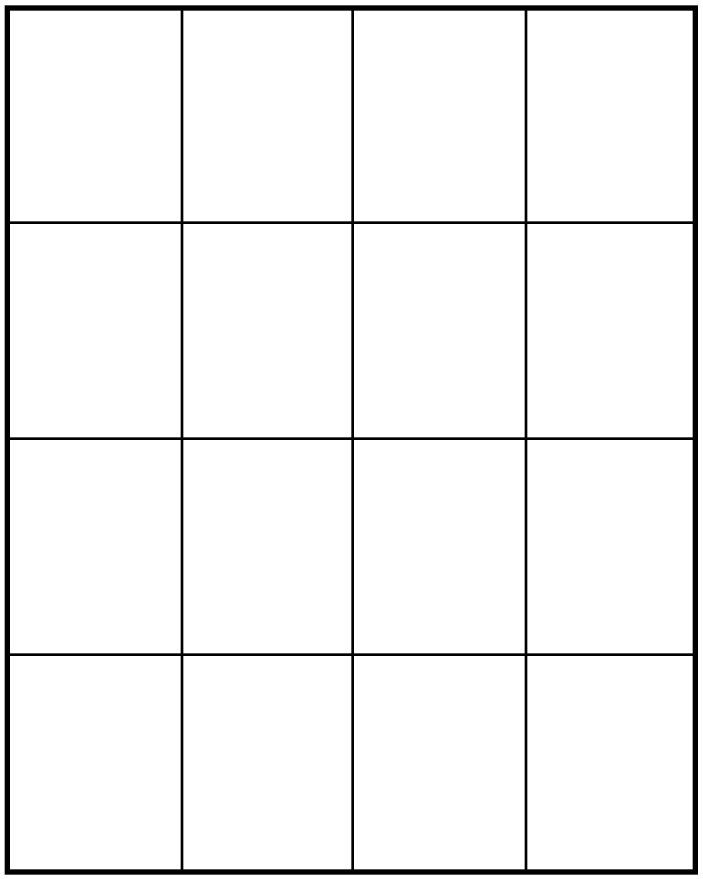
gap extension penalty

because you do not want to add too much of a penalty for extending the gap.

Affine Gap Penalties

- Gap penalties:
  - $-\rho-\sigma$ when there is 1 indel
  - $-\rho-2\sigma$ when there are 2 indels
  - $-\rho-3\sigma$ when there are 3 indels, etc.
  - $-\rho- x\cdot\sigma$ (-gap opening - x gap extensions)
- Somehow reduced penalties (as compared to naïve scoring) are given to runs of horizontal and vertical edges

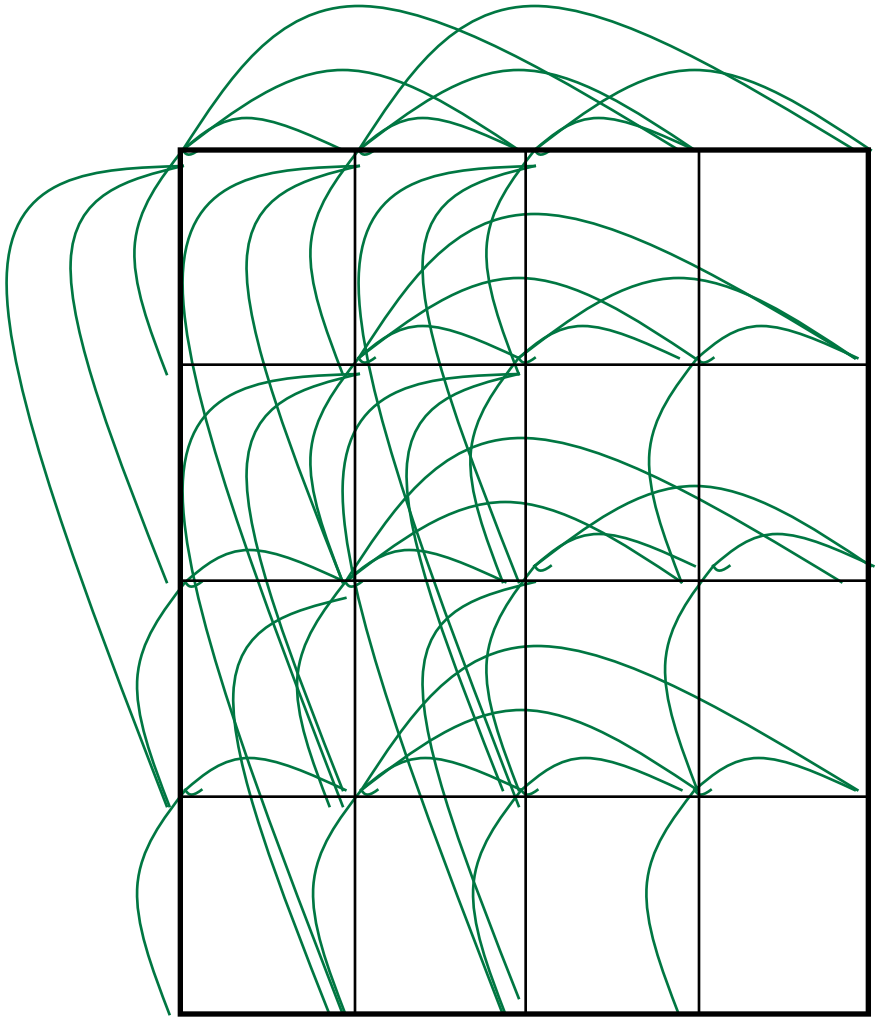
Affine Gap Penalties and Edit Graph



To reflect affine gap penalties we have to add “long” horizontal and vertical edges to the edit graph. Each such edge of length x should have weight

$$-r - x * s$$

Adding "Affine Penalty" Edges to the Edit Graph

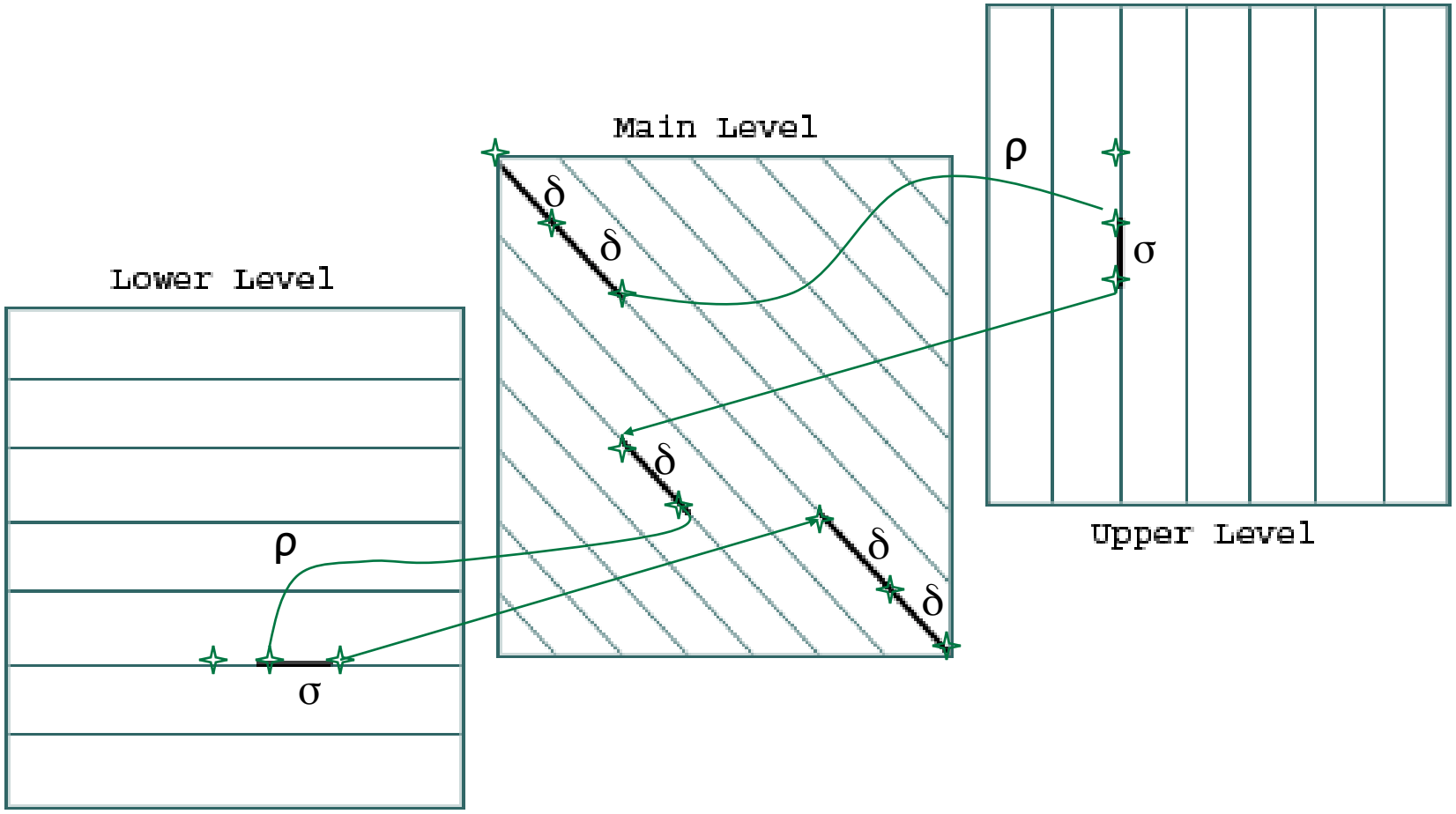


There are many such edges!

Adding them to the graph increases the running time of the alignment algorithm by a factor of ***n*** (where ***n*** is the number of vertices)

So the complexity increases from $O(n^2)$ to $O(n^3)$

Manhattan in 3 Layers



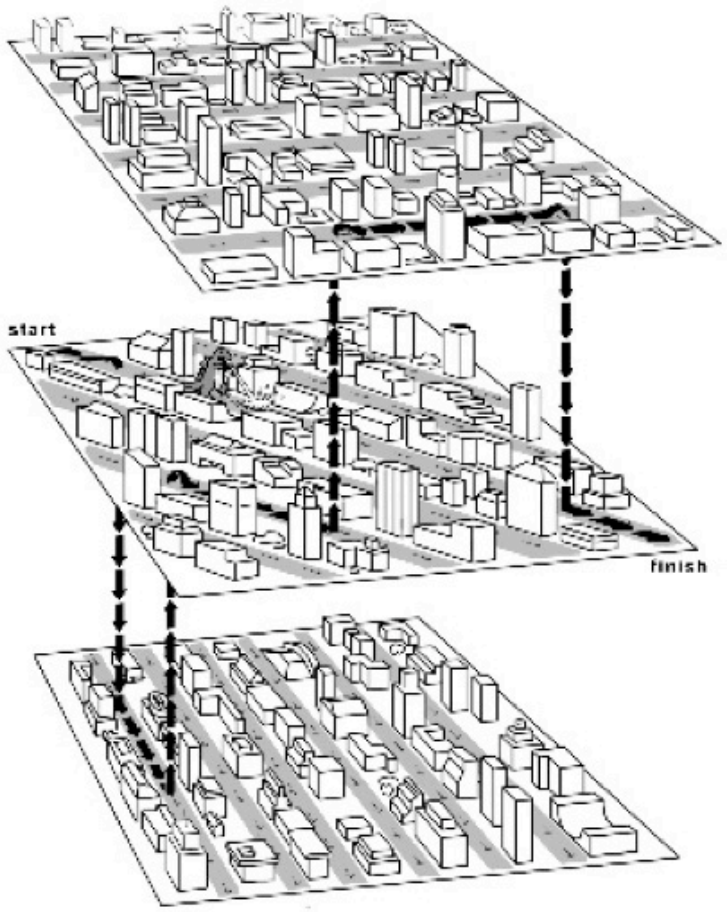
Affine Gap Penalties and 3 Layer Manhattan Grid

- The three recurrences for the scoring algorithm creates a 3-layered graph.
- The top level creates/extends gaps in the sequence *w.*
- The bottom level creates/extends gaps in sequence *v.*
- The middle level extends matches and mismatches.

Switching between 3 Layers

- Levels:
  - The **main level** is for diagonal edges
  - The **lower level** is for horizontal edges
  - The **upper level** is for vertical edges
- A jumping penalty is assigned to moving from the main level to either the upper level or the lower level (-*r*- *s*)
- There is a gap extension penalty for each continuation on a level other than the main level (-*s*)

The 3-leveled Manhattan Grid



Gaps in w

Matches/
Mismatch
es

Gaps in v

Affine Gap Penalty Recurrences

$$\downarrow s_{i,j} = \max \begin{cases} \downarrow s_{i-1,j} - \sigma \\ s_{i-1,j} - (\rho+\sigma) \end{cases}$$

Continue Gap in *w* (deletion)

Start Gap in *w* (deletion): from middle

$$\rightarrow s_{i,j} = \max \begin{cases} \rightarrow s_{i,j-1} - \sigma \\ s_{i,j-1} - (\rho+\sigma) \end{cases}$$

Continue Gap in *v* (insertion)

Start Gap in *v* (insertion):from middle

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(v_i, w_j) \\ \rightarrow s_{i,j} \\ \downarrow s_{i,j} \end{cases}$$

Match or Mismatch

End deletion: from top

End insertion: from bottom